Public Transport Optimization

Phase 5: Project documentation and Submission

Here, we document our project and prepare it for submission.

Tasks:

Documentation:

- *Describe the project's objectives, IoT sensor deployment, platform development, and code implementation.*
- *Include diagrams, schematics, and screenshots of the IoT sensors, transit information platform, and real-time data display.*
- *Explain how the real-time transit information system can improve public transportation services and passenger experience.*

Submission:

- *Share the GitHub repository link containing the project's code and files.*
- *Provide instructions on how to replicate the project, deploy IoT sensors, develop the transit information platform, and integrate them using Python.*
- *Include example outputs of IoT sensor data transmission, platform UI, and real-time data updates*

Public Transport Optimization

1. Project Overview

Introduction

Through the use of IoT (Internet of Things) technology, the "Public Transport Optimisation" initiative seeks to completely transform urban transportation infrastructure. This project aims to enhance the effectiveness, dependability, and general quality of public transportation services by incorporating IoT sensors into public transit vehicles. Clear objectives, a solid IoT sensor system, a platform for real-time transit information, and the integration of these components using Python are the important features of this solution.

Objectives

1. ***Real-Time Data Collection:***

   Install IoT sensors on public transportation vehicles to gather real-time information on passenger counts, locations of individual cars, and other pertinent metrics.

2. ***Data Analysis and Prediction:***

   Create algorithms to analyse the gathered data and forecast arrival timings, service interruptions, and other insights pertaining to transit.

3. ***Real-Time Information Dissemination:***

   Establish a user-friendly public platform or mobile app to offer commuters real-time transit information, such as anticipated arrival times, route updates, and capacity levels.

4. ***Efficiency Enhancement:***

   Reduce wait times and congestion by optimising transit routes and timetables based on data-driven insights.

Key Features

- Vehicle location tracking using Google Maps.
- Real-time passenger count updates.
- Calculation of estimated arrival time.
- Communication between the Android app, server, and Arduino module.

2. Project Components

IOT Sensors

- NEO-6 GPS module
- PIR sensor

Android App

The Android app is responsible for providing users with a user-friendly interface to view the vehicle’s location, passenger count, and estimated arrival time.

Server

The server, developed in Python, acts as an intermediary between the Android app and the Arduino module. It stores data received from the Arduino module and responds to requests from the Android app.

Arduino Module

The Arduino module is responsible for collecting **GPS data from a NEO-6 GPS module** and **passenger count data from a PIR sensor**. It sends this data to the server for storage and retrieval.

3. Directory Structure

The project directory structure is organized as follows:

- /application: Android app source code and resources.
- /server: Python server code and data storage.
- /microcontroller: Arduino code for data collection.
- /docs: Project documentation.

4. Dependencies

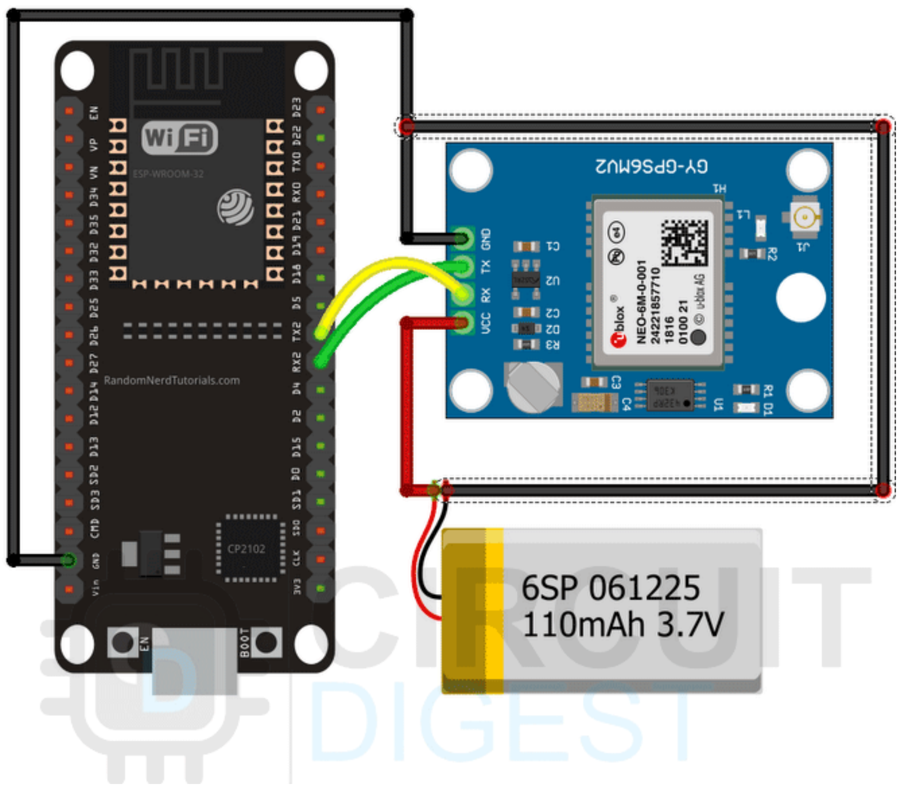
- AndroidX
- Google Play Services (Maps)
- Python 3.6+
- Required Python libraries (specified in /server/requirements.txt)
- Arduino IDE for ESP32 development

5. Server Configuration

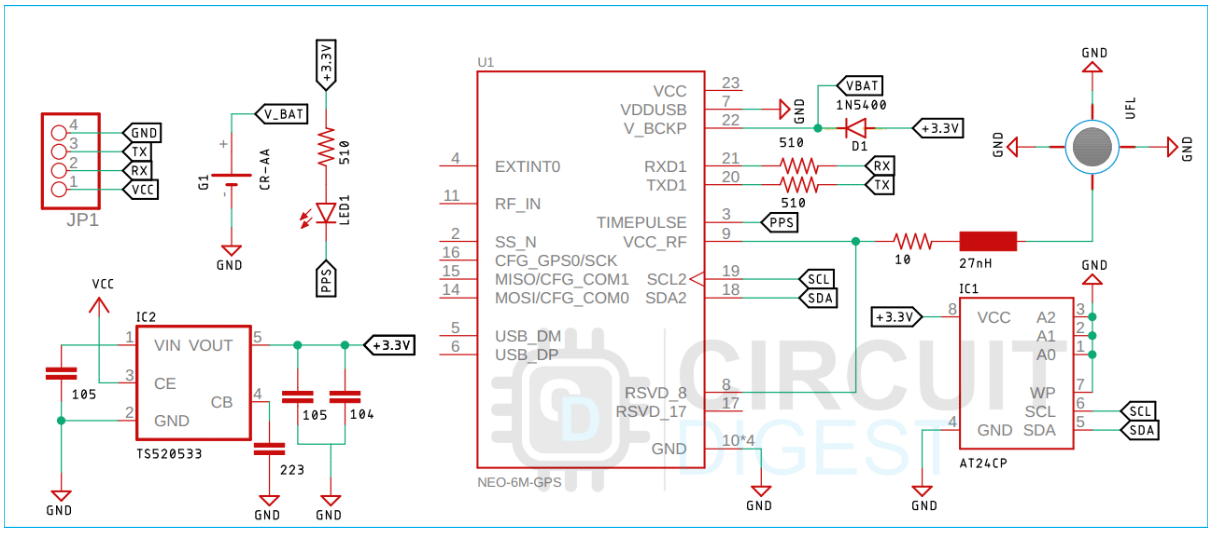
- Our app receives the information of real-time location, ridership data from the server through http requests.
- The RetrieveDataFromServerTask is an AsyncTask that performs an HTTP GET request to a server (specified by the URL) to fetch data related to GPS location and passenger count. The response from the server is expected to be in JSON format.
- In the doInBackground method of RetrieveDataFromServerTask, it sends an HTTP request, retrieves the JSON response, and parses it into a JSONObject. The JSON data is expected to contain latitude, longitude, and passenger count.

6. Setup and Installation

NEO-6M GPS module with ESP32:



Schematic diagram:



7. Code implementation

Microcontroller (Arduino module)

microcontroller.cpp

```
#include <TinyGPS++.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>

#define TX_PIN 2   // TX pin of NEO-6 GPS module
#define RX_PIN 15  // RX pin of NEO-6 GPS module
#define PIR_SENSOR_PIN 12  // Pin for the PIR sensor

const char* ssid = "wifi_ssid";
const char* password = "wifi_password";
const char* serverUrl = "http://127.0.0.1:5000/update_vehicle_data";
";

TinyGPSPlus gps;
int passengerCount = 0;

void setup() {
  Serial.begin(115200);
  Serial1.begin(9600, SERIAL_8N1, RX_PIN, TX_PIN); // Initialize the
GPS module
  pinMode(PIR_SENSOR_PIN, INPUT);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}
void loop() {
  // Check PIR sensor for passenger detection
  if (digitalRead(PIR_SENSOR_PIN) == HIGH) {
    // Passenger boarded
    passengerCount++;
    Serial.println("Passenger boarded. Count: " +
String(passengerCount));
    delay(1000); // Debounce
  }
  while (Serial1.available() > 0) {
    if (gps.encode(Serial1.read())) {
      if (gps.location.isUpdated()) {
        float latitude = gps.location.lat();
        float longitude = gps.location.lng();

        // Send GPS data and passenger count to a server
        sendDataToServer(latitude, longitude, passengerCount);
      }
    }
  }
  delay(10000);  // Update GPS data every 10 seconds
}

void sendDataToServer(float latitude, float longitude, int
passengers) {
  HTTPClient http;

  String data = "lat=" + String(latitude, 6) + "&lng=" +
String(longitude, 6) + "&passengers=" + String(passengers);
  http.begin(serverUrl);
  http.addHeader("Content-Type",
"application/x-www-form-urlencoded");

  int httpResponseCode = http.POST(data);

  if (httpResponseCode > 0) {
    String response = http.getString();
    Serial.println("HTTP Response Code: " +
String(httpResponseCode));
    Serial.println(response);
  } else {
    Serial.println("HTTP Error");
  }

  http.end();
}
```

Server

app_server.py

```
from flask import Flask, request, jsonify
import json

app = Flask(__name)

vehicle_data = {
    "latitude": 0.0,
    "longitude": 0.0,
    "passenger_count": 0
}

@app.route('/update_vehicle_data', methods=['POST'])
def update_vehicle_data():
    data = request.get_json()

    latitude = data.get('latitude')
    longitude = data.get('longitude')
    passenger_count = data.get('passengers')

    vehicle_data["latitude"] = latitude
    vehicle_data["longitude"] = longitude
    vehicle_data["passenger_count"] = passenger_count

    return 'Data received and updated', 200

@app.route('/get_vehicle_data', methods=['GET'])
def get_vehicle_data():
    response = {
        "latitude": vehicle_data["latitude"],
        "longitude": vehicle_data["longitude"],
        "passenger_count": vehicle_data["passenger_count"]
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Application

MainActivity.java

```java
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.content.Context;
import android.content.pm.PackageManager;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import android.os.Bundle;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.concurrent.TimeUnit;
import java.util.List;
import java.util.Arrays;

import com.google.android.gms.maps.model.LatLng;
import com.google.android.libraries.places.api.Places;
import
com.google.android.libraries.places.api.net.FindCurrentPlaceRequest;
import
com.google.android.libraries.places.api.net.FindCurrentPlaceResponse;
import com.google.android.libraries.places.api.net.PlacesClient;
import com.google.android.libraries.places.api.model.Place;
import com.google.android.libraries.places.api.model.PlaceLikelihood;

public class MainActivity extends AppCompatActivity implements
ActivityCompat.OnRequestPermissionsResultCallback {
    private TextView locationTextView;
    private TextView passengerCountTextView;
    private TextView arrivalTimeTextView;
    private PlacesClient placesClient;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        View rootView =
LayoutInflater.from(this).inflate(R.layout.activity_main, null);
        setContentView(rootView);

        locationTextView = rootView.findViewById(R.id.locationTextView);
        passengerCountTextView =
rootView.findViewById(R.id.passengerCountTextView);
        arrivalTimeTextView =
rootView.findViewById(R.id.arrivalTimeTextView);

        // Check and request permissions
        if (checkLocationPermission()) {
            // Permissions are already granted. Proceed with your
location-related tasks.
            initializePlaces();
            new RetrieveDataFromServerTask().execute();
        }
    }

    private boolean checkLocationPermission() {
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            // Permission is not granted. You can request it.
            String[] permissions =
{Manifest.permission.ACCESS_FINE_LOCATION};
            int requestCode = 1;
            ActivityCompat.requestPermissions(this, permissions,
requestCode);
            return false; // Permissions not granted yet.
        } else {
            return true; // Permissions are already granted.
        }
    }

    private void initializePlaces() {
        // Initialize the Places API client
        Places.initialize(getApplicationContext(), "API_KEY");
        placesClient = Places.createClient(this);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);

        if (requestCode == 1) {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                // Permission was granted. You can now proceed with
location-related tasks.
                initializePlaces();
                new RetrieveDataFromServerTask().execute();
            } else {
                // Permission was denied. Handle this situation, e.g., by
showing a message to the user.
                locationTextView.setText("Location permission denied.
Please grant the permission in app settings.");
            }
        }
    }

    private class RetrieveDataFromServerTask extends AsyncTask<Void, Void,
JSONObject> {
        @Override
        protected JSONObject doInBackground(Void... voids) {
            try {
                URL url = new
URL("http://127.0.0.1:5000/update_vehicle_data");
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
                connection.setRequestMethod("GET");

                InputStream inputStream = connection.getInputStream();
                BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
                StringBuilder result = new StringBuilder();
                String line;

                while ((line = reader.readLine()) != null) {
                    result.append(line);
                }

                return new JSONObject(result.toString());
            } catch (IOException | JSONException e) {
                e.printStackTrace();
                return null;
            }
        }

        @Override
        protected void onPostExecute(JSONObject result) {
            if (result != null) {
                try {
                    double latitude = result.getDouble("lat");
                    double longitude = result.getDouble("lng");
                    int passengers = result.getInt("passengers");

                    // Use the Places API to get place name from latitude
and longitude
                    LatLng latLng = new LatLng(latitude, longitude);
                    getPlaceNameFromCoordinates(latLng);
                    // Display the passenger count in the app
                    passengerCountTextView.setText("Passengers: " +
passengers);

                    // Calculate and display the estimated arrival time
                    String estimatedArrivalTime =
calculateEstimatedArrivalTime(latitude, longitude);
                    arrivalTimeTextView.setText("Estimated Arrival Time: "
+ estimatedArrivalTime);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            } else {
                locationTextView.setText("Error retrieving data from the
server.");
                passengerCountTextView.setText("Passenger count not
available.");
            }
        }
    }

    private void getPlaceNameFromCoordinates(LatLng latLng) {
        List<Place.Field> placeFields = Arrays.asList(Place.Field.NAME);

        FindCurrentPlaceRequest request =
FindCurrentPlaceRequest.newInstance(placeFields);


placesClient.findCurrentPlace(request).addOnSuccessListener((response) ->
{
            if (response != null) {
                for (PlaceLikelihood placeLikelihood :
response.getPlaceLikelihoods()) {
                    String placeName =
placeLikelihood.getPlace().getName();
                    updateLocationTextView(placeName);
                    break; // Get the first place as the most likely
                }
            } else {
                updateLocationTextView("Place name not found");
            }
        }).addOnFailureListener((exception) -> {
            updateLocationTextView("Error getting place name");
        });
    }

    private void updateLocationTextView(String placeName) {
        locationTextView.setText("Location: " + placeName);
    }

    private String calculateEstimatedArrivalTime(double latitude, double
longitude) {
        LatLng destination = new LatLng(latitude, longitude);
    String origin = "Starting location";

    // Use the Google Directions API to get estimated travel time
    DirectionsApiRequest directions = new
DirectionsApiRequest(Places.createClient(getApplicationContext()));
    directions.origin(origin);
    directions.destination(destination);
    directions.mode(TravelMode.DRIVING); // You can change the mode as
needed

    try {
        DirectionsResult directionsResult = directions.await();

        if (directionsResult != null) {
            DirectionsRoute route = directionsResult.routes[0];
            DirectionsLeg leg = route.legs[0];

            // Get the estimated duration in minutes
            long durationInMinutes = leg.duration.inSeconds / 60;

            // Calculate arrival time by adding the duration to the
current time
```

```
            Calendar calendar = Calendar.getInstance();
            calendar.add(Calendar.MINUTE, (int) durationInMinutes);

            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm",
Locale.getDefault());
            String estimatedArrivalTime = sdf.format(calendar.getTime());

            arrivalTimeTextView.setText("Estimated Arrival Time: " +
estimatedArrivalTime);
        } else {
            arrivalTimeTextView.setText("Error calculating arrival
time.");
        }
    } catch (ApiException | InterruptedException | IOException e) {
        e.printStackTrace();
        arrivalTimeTextView.setText("Error calculating arrival time.");
    }
    }
    }
}
```

XML Code (UI)

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/locationTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Location: "
        android:textSize="18sp" />

    <TextView
        android:id="@+id/passengerCountTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Passengers: "
        android:textSize="18sp" />

    <TextView
        android:id="@+id/arrivalTimeTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Estimated Arrival Time: "
        android:textSize="18sp" />

</LinearLayout>
```

8. Sample outputs:

From Arduino to server:

```
lat=37.7749&lng=-122.4194&passengers=5
lat=40.7128&lng=-74.0060&passengers=10
lat=37.7749&lng=-122.4194&passengers=6
lat=40.7128&lng=-74.0060&passengers=11
```

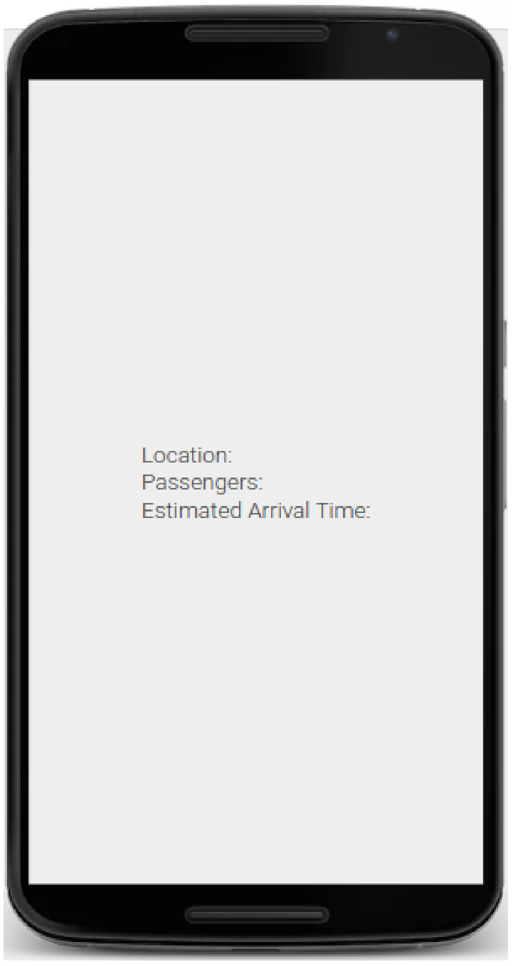
From GPS module to Arduino:

```
Latitude: 37.7749
Longitude: -122.4194
```

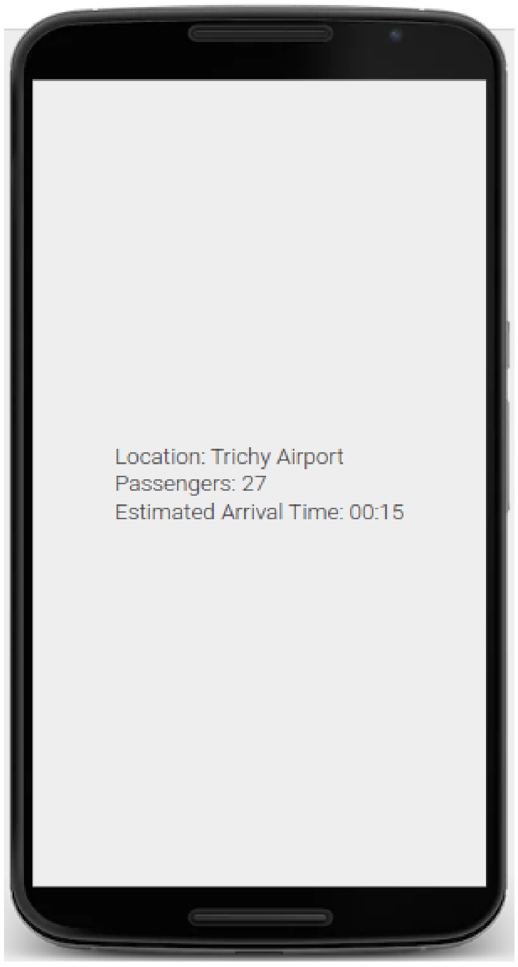
From PIR sensor to Arduino:

```
Passenger Count: 5
```

Platform UI:



Location:
Passengers:
Estimated Arrival Time:



Location: Trichy Airport
Passengers: 27
Estimated Arrival Time: 00:15

Instructions on how to replicate the project, deploy IoT sensors, develop the transit information platform, and integrate them using Python:

1. Configure the environment for development:

Install the software and set up the equipments needed for the project. This includes the GPS module, PIR sensor, Arduino IDE for the Arduino module, Python for the server, and Android Studio for the application.

GPS Module Setup:

NEO-6M GPS module set up with Arduino ESP32.

Wiring Connections:

Connect the GPS module to the ESP32 as follows:

- GPS VCC to ESP32 3.3V
- GPS GND to ESP32 GND
- GPS TX to ESP32 RX (Receive) pin
- GPS RX to ESP32 TX (Transmit) pin

PIR Sensor Setup:

Choose a suitable PIR sensor module for passenger counting. These sensors typically have three pins: VCC, GND, and OUT.

Wiring Connections:

Connect the PIR sensor to the ESP32 as follows:

- PIR VCC to ESP32 3.3V
- PIR GND to ESP32 GND
- PIR OUT to an ESP32 digital input pin (e.g., pin 26)

2. Recognise the Code:

Examine the Arduino module, server, and Android app's code. Examine the interactions and data transfers that occur between these components.

3. Put Dependencies in Place:

Install any dependencies, libraries, or packages that are needed and listed in the project documentation. Make that the versions being used correspond to those listed in the documentation.

4. Set up the server:
Install and configure the Python server according to the project's instructions. Setting up environment variables and verifying the server's operation may be necessary for this.

5. Load the Arduino Code:
Load the Arduino code onto the Arduino module. Ensure that the connections between the GPS module, PIR sensor, and the Arduino are correctly set up.

6. Build the Android App:
Open the Android project in Android Studio and build the app. Make sure you have set up the correct development environment and have connected an Android emulator or device for testing.

7. Test the Project:
Start testing the project to ensure that it functions correctly. Test various scenarios, such as passenger count updates and GPS location tracking.

How the real-time transit information system can improve public transportation services and passenger experience:

Passenger experiences and public transport services can be substantially improved by using a real-time transit information system. Through the provision of real-time updates on timetables, disruptions, and car whereabouts, it affords travellers increased comfort, shorter wait times, and enhanced security. Improved passenger communication, more effective crowd control, and optimised scheduling all benefit transit agencies. The incorporation of real-time data alongside fare payment systems, accessibility features, and environmentally conscious practices enhances the overall quality and sustainability of public transportation. In general, real-time transit information systems help communities and travellers alike by advancing accessibility, effectiveness, safety, and sustainability in urban transportation.